Applying *i** Metrics for the Integration of Goal-Oriented Modeling into MDD Processes

Giovanni Giachetti[1] ggiachetti@pros.upv.es
Fernanda Alencar[2] fmra@ufpe.br
Xavier Franch[3] franch@essi.upc.edu
Oscar Pastor[1] opastor@pros.upv.es

[1] Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain

[2] Universidade Federal de Pernambuco
Av. Acad. Hélio Ramos s/n, 50.740-530, CDU, Recife, Brasil

[3] Universitat Politècnica de Catalunya
C/Jordi Girona, 1-3, 08034 Barcelona, Spain

January 2010

Applying *i** Metrics for the Integration of Goal-Oriented Modeling into MDD Processes

Giovanni Giachetti[1], Fernanda Alencar[2], Xavier Franch[3], Oscar Pastor[1]

[1]Centro de Investigación en Métodos de Producción de Software
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain
{ggiachetti, opastor}@pros.upv.es
[2]Universidade Federal de Pernambuco
Av. Acad. Hélio Ramos s/n, 50.740-530, CDU, Recife, Brasil
fmra@ufpe.br
[3]Universitat Politècnica de Catalunya (UPC)
UPC-Campus Nord, Omega-122, 08034 Barcelona, Spain
franch@lsi.upc.edu

Abstract. Nowadays, there exist modeling techniques that provide good support for the requirements elicitation and analysis of complex scenarios, such as the *i** modeling framework. However, the application of these requirements models into Model-Driven Development (MDD) processes is still dependent on the experience of analysts and designers to manually transform the defined requirements models into an appropriate MDD model. Certain approaches have proposed guidelines to facilitate and partially automate this translation, but there is a lack of validation rules establishing how to build *i** models for an improved generation of the corresponding MDD models. Thus, in this paper, we propose a set of metrics that are oriented to validating the adequacy of *i** models as the starting point for MDD processes, as well as a process for the application of the proposed metrics in the *i** framework.

Keywords: *i**, MDD, Metrics, Validation, UML Profile.

1 Introduction

The present software development context is rapidly moving to the Model-Driven Development (MDD) paradigm [26], which has motivated the emergence of multiple MDD approaches oriented to automating the final software product generation by means of model compilation processes. Just as any software development process does, MDD processes also require an appropriate requirements elicitation to obtain software products that fit the customers' needs. For this requirement elicitation, we can find model-oriented approaches that provide suitable alternatives for the analysis of complex scenarios. This is the case of the *i** framework, which is a goal-oriented approach used in the object- and agent-oriented worlds. However, there is still a gap between analysis approaches of this kind and the effective application of MDD

processes, since requirement models, such as *i** models, are not oriented to automatic software generation. Therefore, to apply *i** models in a MDD process, it is necessary to transform them into an appropriate input for the model compilation process. In other words, to transform the *i** analysis model into a MDD-oriented model.

Certain approaches have defined guidelines to perform this transformation [1][2][15], but, in general terms, these guidelines consider *i** models to already be perfectly defined for the MDD model generation. We know that in real application contexts this idealist scenario is not feasible, and hence, additional validation mechanisms are required to assure that the defined *i** models are valid for the generation of the corresponding MDD models.

Therefore, this paper presents a set of metrics oriented to validating *i** models for their application in a particular MDD approach, which is centered in the use of UML-like class models. Since the target of the proposed metrics is a class model generation, these can be used as reference by other MDD approaches that use class models for the elaboration of their conceptual models. The particular MDD approach considered in this paper is the OO-method approach [24], which has been successfully applied to industrial software development [25]. The proposed metrics have been defined taking as reference the set of transformation guidelines presented in [1] for the transformation of *i** models into the corresponding OO-Method class models.

However, we understand that MDD approaches have particular characteristics related to their model compilation processes and application domains, and hence, they may require specific validation metrics that must be aligned to their modeling needs. For this reason, we also present a general process for the integration of any required metric into the *i** framework in order to provide a tool that can be used by different MDD approaches to integrate their particular validation metrics and to facilitate the use of *i** into different MDD processes. This integration process is also framed within the MDD context and takes advantage of the existing standards and technologies for the specification of modeling languages and model transformation.

The rest of the paper is organized as follows. Section 2 shows the background related to metrics for *i** and MDD. Section 3 introduces the proposed *i** metrics for MDD integration. Section 4 details the process to integrate the proposed metrics into the *i** framework. Section 5 presents the application of the proposed metrics into a brief *i** example. Finally, Section 6 presents our conclusions and further work.

2 Background

In this section, we discuss the applicability of *i** in MDD processes by considering our previous experience in the generation of MDD conceptual models from *i** models. In addition, the relevance of the use of metrics to validate the *i** and MDD integration is also discussed to clarify the motivation of this paper.

2.1 Transforming *i** models into MDD conceptual models

Currently, the integration of requirements modeling into MDD processes looks to be the next natural step to obtaining a sound and comprehensive development process

that covers the full software life-cycle, aligning the generated software product with the requirements specification. Certain proposals have established guidelines to transform requirements models into MDD-oriented conceptual models [1][15], with *i** being one of the requirements modeling frameworks chosen for this purpose. The reason why *i** models must be transformed into MDD conceptual models is related to the nature of these two kinds of models: while *i** models are centered on organizational analysis and not on the software representation, MDD conceptual models are centered on the precise representation of software systems, thus allowing the generation of the final software products.

In [1], we showed that it is possible to partially infer MDD models from *i** models. To do this, we propose the MDD model generation by identifying the organizational process to be automated from the *i** model. Next, the *i** elements involved in this process are transformed into one or more elements of a class model that is compliant to the OO-Method approach [25] by applying a set of transformation guidelines. As a result, after the transformation of the *i** model, an initial class diagram is obtained. This diagram must be refined (at design time) to obtain a complete OO-Method class model that is used as the starting point for the automatic generation of the intended system. For example, it is necessary to specify the multiplicity of the associations among classes because this information cannot be inferred from the *i** model.

In this context, it is important to determine if the defined *i** models provide a proper specification to perform the transformation process. It is precisely at this point where the specification of metrics to validate the *i** models becomes necessary.

2.2 Validation of *i** Models for MDD Processes Using Metrics

In the literature, there exist different works related to the definition of *i** metrics such as [10] and [9]. Generally speaking, these works propose a general framework that addresses the validation of the correct specification of *i** models by centering their attention on the correct problem analysis and requirements elicitation. The framework has been customized for certain contexts, e.g. business process assessment [9].

From the MDD side, several works have proposed metrics to validate object-oriented models [11][27], which is the most widely used conceptual representation for MDD approaches. Additionally, there exist particular validation metrics that are oriented to the correct compilation of the defined MDD-oriented conceptual models, such as [17].

Nevertheless, there is a lack of metrics that are oriented to validate the specification of *i** models for MDD purposes. In this paper, we precisely want to tackle this issue by providing a set of validation metrics for *i** models used in MDD processes. For the definition and application of these metrics, we have considered existent standards and modeling technologies in order to obtain a MDD-oriented validation solution, which facilitates the application of metrics for different MDD approaches. The technologies and standards involved are: metrics specification approaches [4][5], the *i** framework [13] for requirements modeling, *i** metrics definition approaches [9][10], OMG Standards [19][20][21][22] for metamodeling and model extensions definition, ATL [6] for model transformations, and Eclipse Model Development Tools [7] for tool support.

3 A Suite of *i** Metrics for MDD

This section introduces the metrics proposed in this paper to validate *i** models that are used as the starting point of MDD processes. These metrics are based on the issues that have arisen in the application of the guidelines presented in [1], which are centered on generating a specific MDD class model from an *i** model and whose rationale is not discussed in this paper.

First, it is necessary to determine what type of *i** constructs are considered for the MDD conceptual model generation. At the current stage of our transformation proposal, neither goals nor softgoals are translated. Goals are not translated because they do not enclose any information needed for MDD conceptual model generation. Softgoals are not translated because they mainly express non-functional requirements, which are currently not considered in [1] (although they may be in the future). Therefore, actors, resources and tasks are the relevant *i** constructs that need to be addressed in this paper.

Table 1 shows a summary of the reference transformation guidelines that are used for the formulation and exemplification of the proposed metrics, which correspond to a subset of the guidelines presented in [1]. This table shows the *i** constructs involved in the transformation, the additional information that must be considered to perform the transformation, and the target constructs of the class model. It is important to note that we call to the tasks related to the dependee and depender actors in a dependency relationship as *dependee task* and *depender task*, respectively.

Table 1. Guidelines for the transformation of *i** models into OO-Method class models.

i* **Construct**	**Additional Information**	**Class Model Construct**
Actor		Class
Resource	Physical entity	Class
	Informational entity related to a physical resource or an actor	An attribute that represents information of the class generated from the actor or physical resource
	Resource in a decomposition tree	Input arguments for the service generated from the related task
	Dependum resource	Input argument of the depender task
	Physical entity inside of an actor boundary	An association between the classes generated from the physical resource and the owner actor
Task	Participating in a resource dependency as depender or dependee	A service of the class generated from the dependum resource
	If generates a resource	A creation service of the class generated from the resource
Dependency link	Where the *dependum* resource and the *depender* and *dependee* actors are transformed in classes	Associations are automatically defined among the generated classes

The presented guidelines are combinable, for example, a physical resource that is also a dependum resource generates a class in the class model and also generates an input argument of the service that is generated from the depender task. It is also important to remark that just tasks and resources (not actors) require additional modeling information for the correct generation of the corresponding class model constructs (for instance, classifying a resource as *informational* or *physical*), which may not be present in the original *i** framework; therefore, our metrics will focus on

these elements. Section 5 shows an example *i** model and its transformation according to the rules defined in Table 1.

For the formulation and rationale of the proposed metrics, we have considered the Goal-Question-Metric (GQM) approach [4]. Figure 1 shows the results obtained in the GQM application. The derived questions focus on the identification of resources and tasks that are wrongly defined or can be improved for the class model generation.

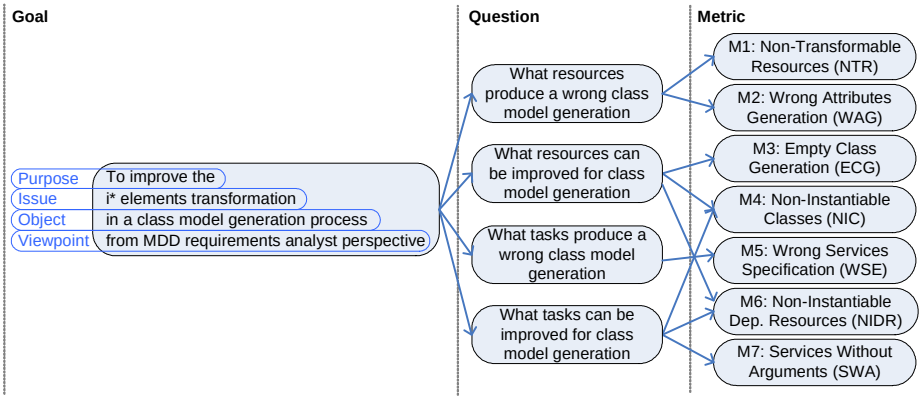

Figure 1. Application of GQM for definition of proposed metrics.

From this GQM application, the metrics for the validation of *i** elements are obtained. However, since the validations required by the OO-method approach are probably different than those required by other specific MDD approaches, only those metrics that can be generically applied to different object-oriented MDD approaches are presented in this paper. These metrics are dived into two groups by distinguishing the effect of the validation over the generated class model.

3.1 Metrics to validate the generation of classes and attributes

M1. Non-Transformable Resources (NTR). This metric considers the identification of resources that are not specified as physical or informational entities, since the transformation of *i** resources varies according to the kind of resource involved. Thus, those resources that are selected to generate the class models must be categorized into one of these two kinds (physical or informational), if not, the generation of the corresponding class model constructs cannot be performed.

M2. Wrong Attributes Generation (WAG). This metric considers the identification of informational resources that are not related to a physical resource or to an actor. As the table of transformation guidelines shows (Table 1), the informational resources are involved in the generation of class attributes. Therefore, these resources must be related to an actor or a physical resource (transformed into classes in the class model) for their correct generation; otherwise, it is impossible to transform these resources into attributes without having a class to contain them. This metric is very valuable for the identification of the physical entities that participate in the final software product.

M3. Empty Class Generation (ECG). This metric considers the identification of physical resources or actors that do not have related informational resources. In the class model generation, the informational resources correspond to attributes of the classes generated from physical resources or actors. Therefore, an *i** resource or an actor without related informational resources means that a class without attributes will be generated in the class model, which is not valid for concrete classes (in OO-method abstract classes cannot be defined). This metric helps in the appropriate identification of informational resources in the *i** model.

3.2 Metrics to validate the generation of services

M4. Non-Instantiable Classes (NIC). This metric considers the identification of physical resources that do not have a task related to their production. In a class model for software generation, a relevant element that must be identified is the service that produces new instances of a class, since without this service the class is not properly defined (all the defined classes must be capable of generating their instances), and hence, the correct software product cannot be generated. According to the transformation guidelines, these kinds of services are identified from the *i** tasks that produce physical resources. Therefore, if there are resources without related tasks of this kind, in the resultant class model, there will be classes without creation instance services. For those "non-instantiable" resources, the generation of the corresponding class model constructs is still possible by means of the generation of default services for the creation of instances. Thus, this validation is not only oriented to providing additional modeling information for the correct identification of the tasks that produce resources, but also to determining if additional tasks for production of the defined resources are required. It is important to mention that actors are not validated by this metric (even though actors are also transformed into classes) because actors are considered as pre-existent entities in the context of the analyzed problem. Hence, the definition of tasks that produce actors in the *i** model is not common. Therefore, the actors are directly transformed into classes with a default instance creation service.

M5. Wrong Services Specification (WSE). This metric considers the identification of dependee tasks that produce a resource that is different from the related dependum resource. According to the transformation guidelines (Table 1), the dependee and depender tasks in a resource dependency relationship are transformed into services of the class generated from the dependum resource. Therefore, these tasks cannot participate in the generation of another physical resource (that is different from the dependum resource), since these tasks are transformed into class services and an instance creation service can only produce an instance of its owner class.

M6. Non-Instantiable Dependum Resources (NIDR). This metric considers the identification of dependum physical resources that do not have a related production task. In a resource dependency relationship, the dependee task is the responsible for providing the resource that is required by the depender task. In this context, the dependee task has a high probability of being the task that produces (creates) the

required resource (dependum). Therefore, it is important to identify those dependum physical resources that do not have a related production task in order to verify if the related dependee tasks are correctly marked for the software generation. It is also important to indicate if these dependee tasks are responsible for the dependum resource production. This metric is relevant for the generation of the correspondent instance creation services in the class model.

M7. Services Without Arguments (SWA). This metric considers the identification of the final tasks in a decomposition tree that have no related resources. The resources related to a task allow the identification of the arguments related to a service (generated from the involved task) in the resultant class model. Therefore, for this metric, we consider those tasks that have the following properties: 1) they are leaves in a decomposition tree; 2) neither tasks nor resources appear in their decomposition; 3) they are not a dependers in resource dependencies (i.e., they are not related to a dependum resource); and 4) they do not produce resources since the instance creation services (that are generated from tasks that produce resources) have the properties of the owner class as arguments.

For those tasks that hold with the previous conditions, it is impossible to automatically determine the related arguments when the corresponding services of the class model are generated. It is important to clarify that all services (except instance creation services) have at least one default argument, which is the reference to the object that executes the service. Therefore, with this metric we are trying to identify those arguments that differ from this default argument.

In the proposed metrics, it can be observed that there are no metrics related to the generation of relationships among classes since, according to the transformation guidelines presented in [1], relationship generation directly depends on the correct definition of physical resources which are already considered in the metrics proposed. However, particular MDD approaches may be interested in defining metrics for relationship generation (or another new construct). Thus, for the integration of the proposed metrics and new metrics defined for specific MDD approaches, we propose the process presented in the next section.

4 Integration of Validation Metrics into the *i** Framework

This section explain how the proposed metrics are integrated into the *i** framework by following the process presented in Figure 2. For the elaboration of this process, we have considered the method for definition of *i** metrics presented in [9], and the *i** metrics patterns presented in [10]. The involved steps are described below.

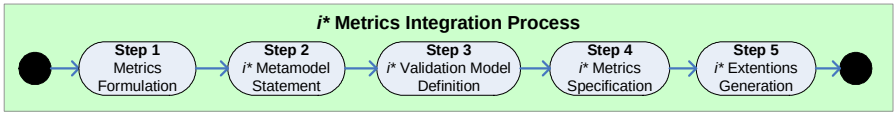

Figure 2. *i** Metrics Integration Process.

Step 1: Metrics Formulation. The first step of the process considers the appropriate formulation of the validation metrics. This means identifying the *i** elements that participate in the MDD conceptual model generation, and, from these, identifying the aspects that must be validated to assure a correct *i** model transformation. Next, by following a metrics formulation approach, such as GQM, the corresponding metrics are defined. This step of the process was performed in Section 3.

Step 2: *i Metamodel Statement.** The second step corresponds to stating the target *i** metamodel. For this paper, we have used the EMOF [19] *i** Metamodel presented in Figure 3. This figure only shows the structural representation of the reference *i** metamodel; additional features such as derived values, constraints, and operations have been omitted to simplify the metamodel representation. For the elaboration of this *i** metamodel, proposals such as [3] and [16] were considered.

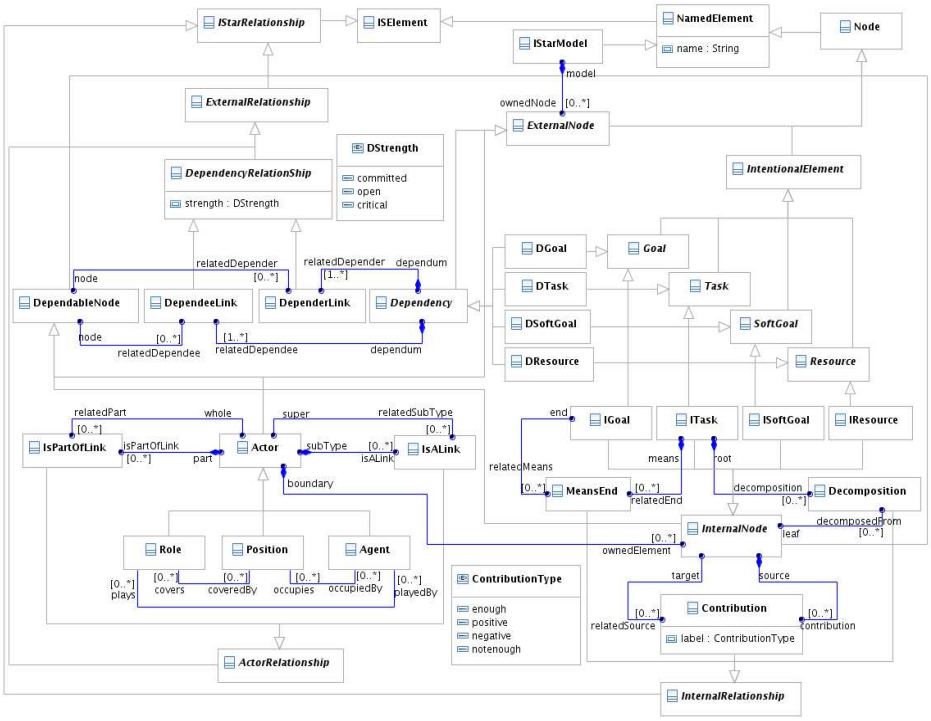

Figure 3. *i** Metamodel.

Step 3: *i Validation Model Definition.** The third step of the process consists in the definition of a validation model (see Figure 4), which is defined according to the EMOF standard [19], the same as the presented *i** metamodel. This validation model includes the information required for the correct application of the metrics. Specifically, it must include those elements that are not present in the reference *i** metamodel. This modeling information is also relevant for the correct generation of the corresponding MDD class models according to the transformation guidelines

presented in Table 1. Figure 4 also shows the mapping information that indicates the correspondences among the elements of the validation model and the *i** metamodel.

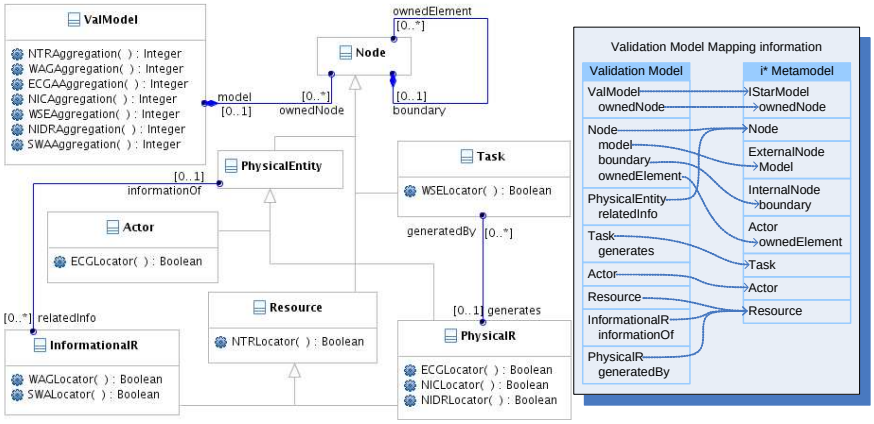

Figure 4. Validation Model.

Step 4: *i Metrics Specification.** The fourth step of the process corresponds to the OCL specification of the metrics, which must be included in the validation model. This specification is performed by considering the *i** metamodel and the validation model. Figure 4 shows the names and outputs of the different OCL rules defined in Table 2. For the metrics specification, we have applied the metrics patterns presented in [10], specifically, the *aggregation*, *locator*, and *condition-checker* patterns. The *locator* and *condition-checker* patterns are used to identify the elements involved in the metrics evaluation, and the *aggregation* pattern is used to return the final numerical value. Additional transformation patterns can be applied later to the results of the metrics to produce different model analyses, for instance, the factor of non-instantiable dependum resources in relation to the total amount of dependum resources defined in the model (from the metric NIDR). A very useful aspect of the application of these patterns is that the *i** elements that must be fixed to improve the class model generation can be easily identified by means of the *locator* pattern.

In addition, since the proposed metrics are for validation purposes, we have introduced a new property in the metrics specification, which corresponds to the alert levels that are related to the defined metrics, these are: 1) Critical: indicates that the situation identified by the metric prevents the transformation of the involved i* elements. 2) Warning: indicates that there is a modeling issue that can be fixed to improve the class model generation. 3) Information, indicates that it is possible to add additional information to the *i** model to improve the class model generation process.

Step 5: *i Extensions Generation.** Finally, in the fifth step of the process, the validation model and the OCL specification of the metrics are used to generated the extensions in order to integrate the proposed metrics into the *i** framework (see Figure 5). These extensions are generated by means of the proposal presented in [12] for automatic UML profile generation, which uses the mapping information presented in Figure 4.

Table 2. Metrics specification in the OCL language.

Metric	Subject of Measure	Alert Level
M1: Non-Transformable Resources (NTR)	*i** Resources	Critical

```
Context: Model::NTRAggregation() : Integer
Body: result = self.ownedNode->select(rs|rs.oclIsKindOf(Resource) and
      rs.NTRLocator())->size() + self.ownedNode.ownedElement
      ->select(rs|rs.oclIsKindOf(Resource) and rs.NTRLocator())->size()

Context: Resource::NTRLocator() : Boolean
Body: result = (not self.oclIsKindOf(PhysicalResource)
      and not self->oclIsKindOf(InformationalResource))
```

M2: Wrong Attributes Generation (WAG)	*i** Informational Resources	Critical

```
Context: Model::WAGAggregation() : Integer
Body: result = self.ownedNode->select(irs|irs.oclIsKindOf(InformationalResource)
      and irs.WAGLocator())->size() + self.ownedNode.ownedElement->select(irs|
      irs.oclIsKindOf(InformationalResource) and irs.WAGLocator())->size()

Context: InformationalResource::WAGLocator() : Boolean
Body: result = self.informationOf->isEmpty()
```

M3: Empty Class Generation (ECG)	*i** Physical Resources	Information

```
Context: Model::ECGAggregation() : Integer
Body: result = self.ownedNode->select(act|act.oclIsKindOf(Actor) and
    act.ECGLocator())->size() + self.ownedNode->select(prs|prs.oclIsKindOf
   (PhysicalResource) and prs.ECGLocator())->size() + self.ownedNode.ownedElement
    ->select(prs|prs.oclIsKindOf(PhysicalResource) and prs.ECGLocator())->size()

Context: Actor::ECGLocator() : Boolean
Body: result = self.relatedInfo->isEmpty()

Context: PhysicalResource::ECGLocator() : Boolean
Body: result = self.relatedInfo->isEmpty()
```

M4: Non-Instantiable Classes (NIC)	*i** Physical Resources	Information

```
Context: Model::NICAggregation() : Integer
Body: result = self.ownedNode->select(prs|prs.oclIsKindOf(PhysicalResource)  and
      prs.NICLocator())->size() + self.ownedNode.ownedElement->select(prs|
      prs.oclIsKindOf(PhysicalResource) and prs.NICLocator())->size()

Context: PhysicalResource::NICLocator() : Boolean
Body:    result = self.generatedBy->isEmpty()
```

M5: Wrong Services Specification (WSE)	*i** Dependee Tasks	Critical

```
Context: Model::WSEAggregation() : Integer
Body: result = self.ownedNode.ownedElement->
      select(it|it.oclIsKindOf(ITask) and it.WSELocator())->size()

Context: Task::WSELocator() : Boolean
Body: result = self.oclIsTypeOf(DependableNode)
      and not self.generates->isEmpty() and not self.dependeeLink->isEmpty()
      and self.generates <> it.dependeeLink.dependum
```

M6: Non-Instanciable Dependum Resources (NIDR)	*i** Dependum Physical Resources	Warning

```
Context: Model::NIDRAggregation() : Integer
Body: result = self.ownedNode->select(dprs|dprs.oclIsKindOf(PhysicalResource) and
      dprs->oclIsKindOf(DResource) and dprs.NIDRLocator())->size()

Context: PhysicalResource::NIDRLocator() : Boolean
Body:    result = self.oclIsKindOf(DResource) and self.generatedBy->isEmpty()
```

M7: Services Without Arguments (SWA)	*i** Tasks	Warning

```
Context: Model::SWAAggregation() : Integer
Body: result = self.ownedNode.ownedElement->
      select(it|it.oclIsKindOf(ITask) and it.SWALocator())->size()

Context: ITask::SWALocator() : Boolean
Body:    result = self.decomposition->isEmpty()                 /* condition 1 */
         or self.decomposition->select(e|e.oclIsTypeOf(Resource)/* condition 2 */
         or e.oclIsTypeOf(Task))->isEmpty()
         or self.dependerLink->isEmpty()                        /* condition 3 */
         or self.generates->isEmpty()                           /* condition 4 */
```

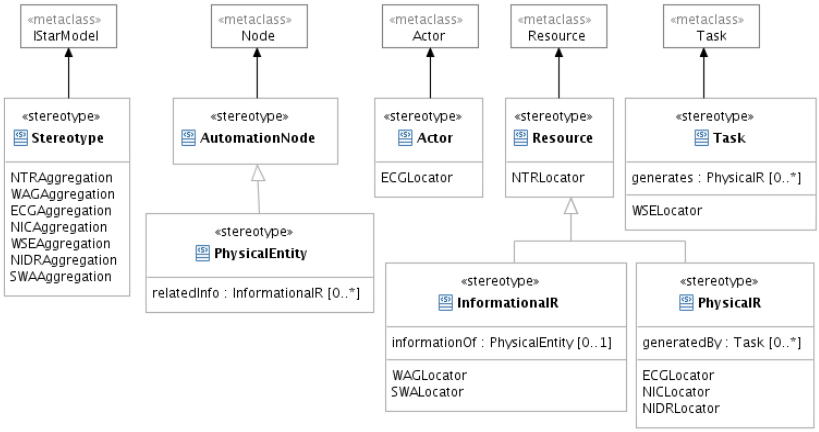

Figure 5. UML Profile to extend the *i** metamodel with the proposed metrics.

It is important to mention that, in contrast to what its name suggest, a UML profile can be used to extend any MOF-based metamodel (not only the UML metamodel only) such as the *i** metamodel used in this paper. In addition, the UML profile is a lightweight extension mechanism that does not change the target metamodel, and it has a standardized definition [21] and interchange format [23]. Therefore, it is a suitable alternative for the application of our proposal.

In the generated UML profile, the parts of the OCL specification of the metrics that are written in italics must be changed according to the defined stereotypes and tagged values. For instance, the specification for metric *M2* (Wrong Attributes Generation) is finally defined as follows:

```
Context: ValModel::NTRAggregation() : Integer
Body:    result = self.ownedNode->select(rs|
                  rs->isStereotyped(Resource) and rs->NTRLocator())->size() +
                  self.ownedNode.ownedElement->select(rs|
                  rs->isStereotyped(Resource) and rs->NTRLocator())->size()

Context: Resource::NTRLocator() : Boolean
Body:    result = not self.isStereotyped(PhysicalR)
                  and not self->isStereotyped(InformationalR)
```

In the resultant OCL specification, it can be observed that the generated stereotypes are used for the identification of the different kinds of resources. In addition, the operation *oclIsTypeOf* is replaced by the operation *isStereotyped*, since the validation is based on the generated extensions. It is important to mention that the OCL operation *isStereotype* is not part of the OMG specification and is only used to simplify the metric's specification. Thus, for the application of the proposed metrics, this operation must be defined or implemented according to the OCL interpreter used. For instance, in ATL this operation must be implemented as follows:

```
helper context UML!Element def : isStereotyped(name:String):Boolean =
not self.getAppliedStereotypes()->select(s|s.name=name)->isEmpty();
```

5 Applying the Proposed Metrics

This section exemplifies how the proposed *i** metrics are used to validate and improve the generation of the corresponding class model. To do this, we present a brief example *i** model (see Figure 6) that is defined from the OO-Method case study presented in [18], which is related to the operation of a Photography Agency. The model includes stereotypes and tagged values that are related to the generated profile (see Figure 5). These are needed for the *i** transformation process. This case study is also used in [1] for the presentation of the reference *i** transformation guidelines. In particular, the presented *i** model shows the reception of work requests (i.e. job applications) from photographers that want to be hired. Due to space constraints, only a simplified version of the complete case study is presented. It is important to mention that, in the complete *i** model, not all the *i** elements are involved in the transformation process. Only those elements that are related to the intended system are considered (i.e. the involved actors). These elements are the stereotyped elements.

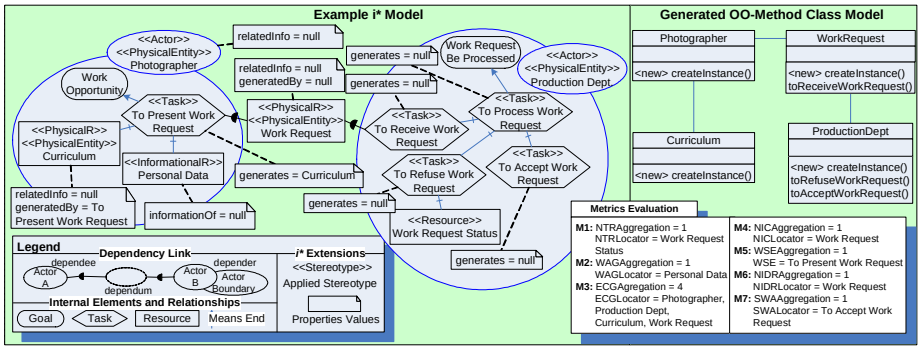

Figure 6. *i** Model Transformation Example.

Figure 6 also shows the results obtained from the metrics evaluation. These are: 1) the values obtained by each OCL rule defined in the UML profile by indicating: the final return numerical value for the rules related to the *aggregation* pattern; and 2) the *i** elements that return *true* for evaluation of OCL rules related to the *locator* pattern. The figure also shows the class model generated from the *i** model applying the transformation guidelines shown in Table 1.

From the metrics evaluation, it can be observed that the *i** elements that cannot be transformed (the resources *Work Request Status* and *Personal Data*, and the task *To Present Work Request*) are identified by critical validation metrics (metrics NTR, WAG, and WSE, respectively). Thus, if we consider the results obtained from these validation metrics, it is possible to solve the issues related to the identified elements before performing the *i** model transformation. For instance, the metric *NTR* indicates the need to state whether the resource *Work Request Status* corresponds to a *physical* resource or to an *informational* resource.

In the metrics evaluation, it can be observed that the warning metric M6 (NIDR) indicates that the involved dependum resource does not have a related production task. In this case, according to the metric definition (see Section 3), the related dependee task *To Present Work Request* has a high probability of being the

production task of this dependum resource. In addition, the critical metric *WSE* indicates that the task *To Present Work Request* is in fact a production task but is wrongly related to the resource *Curriculum*. Thus, we have considered that the task *To Present Work Request* is the production task for the resource *Work Request*. Figure 7 shows the *i** model obtained after the corrections and improvements related to the identified metrics.

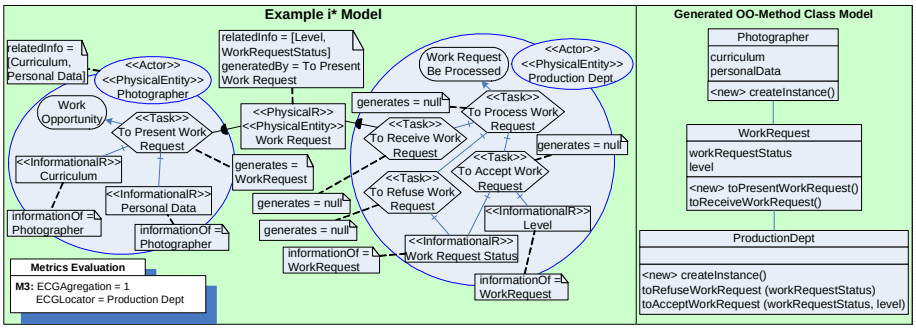

Figure 7. Improved *i** Model Transformation Example.

In Figure 7, it can be observed that most of the informational metrics have also been considered to improve the *i** model transformation. For instance, from the metric *SWA*, which indicates the lack of resources in the decomposition of the task *To Accept Work Request*, it was possible to identify that this task is decomposed into the resources *Work Request Status* and *Level*. These two resources are transformed into input arguments of the service generated from the task involved. Also, since these two resources are specified as informational resources, they generate attributes for the related physical resource *Work Request*. Thus, in the resultant *i** model, there is only one informational metric that has not been considered for improvements. However, since this metric is only informational, it does not prevent the correct generation of the class model. This demonstrates the importance of indicating the alert levels.

Finally, as Figure 7 shows, the classes of the class model obtained from the improved i* model are correctly generated, since the class curriculum generated from the previous *i** model (Figure 6) actually corresponds to an attribute of the class *Photographer* (situation identified with the metric *ECG*). The argument specification of the generated services is also improved and instance creation services are identified. Additionally, a better attribute generation is obtained for the classes *Photographer* and *Work Request*.

6 Conclusions and Further Work

The integration of *i** models for requirements elicitation in MDD process is a step beyond going from Model-Driven Development (MDD) to Model-Driven Engineering (MDE) [14], where different modeling approaches can be integrated to obtain improved software products. This paper has presented new results in this direction by adding correctness conditions formulated as *i** metrics to an existing

method proposed in [1]. This paper has also presented a process for the integration of the proposed metrics into the *i** framework, which takes the metrics definition framework presented in [9][10] as reference. Thus, different MDD approaches can use this integration process to apply their particular metrics into the *i** framework.

One of the main advantages of the proposed metric integration process is that the entire metrics specification is performed by following the model-driven philosophy, where the metrics and the required modeling information are specified in a validation model by using metamodeling standards. Additionally, the extensions over the *i** framework are defined by means of standardized lightweight extension mechanisms that do not alter the original *i** metamodel specification, which permits the compatibility with existent technologies that use the same metamodel as reference. Thus, the proposed integration process solves the difficulties of defining metrics that require additional modeling information without altering the original *i** specification.

For the application of the proposed metrics, we did not find tools that provided transparent support for all the modeling features considered, and hence, additional programming effort was necessary. However, the current development of tools that provide support to the standards considered (such as the Eclipse UML2 project [8] used in this paper) and the increasing number of research works that take advantage of these technologies are indictors that improved tools for the transparent support of these standards will appear in a not-too-distant future. This also motivates the emergence of new approaches for integration and validation oriented to improving the MDD capabilities and the quality of the generated software products.

Further work considers the analysis and elaboration of new validation metrics to improve the application of *i** models within MDD processes as well as the development of empirical studies to obtain result about the benefits of using requirements models in MDD processes. In addition, we are working on a guide that shows how to solve the different implementation and configuration issues in order to apply the presented proposal with existent open-source tools. Finally, we are preparing the publication of the complete *i** metamodel used in this article, which is defined according to the EMOF metamodeling capabilities using open-source technologies [7]. Thus, this *i** metamodel can be used as reference for the construction of interoperable *i** tools.

Acknowledgments. This work has been developed with the support of MEC under the projects SESAMO TIN2007-62894 and ADICT TIN2007-64753 and cofinanced by FEDER. Also, it is partially supported by CAPES.

References

1. Alencar, F., Marín, B., Giachetti, G., Pastor, O., Castro, J., Pimentel, J.H.: From i* Requirements Models to Conceptual Models of a Model Driven Development Process. In: 2nd Working Conference on The Practice of Enterprise Modeling (PoEM). Springer LNIBP (2009)
2. Alencar, F.M.R., Pedroza, F.P., Castro, J., Amorim, R.C.O.: New Mechanisms for the Integration of Organizational Requirements and Object Oriented Modeling. In: 6th Workshop on Requirements Engineering (WER'03) (2003)

3. Ayala, C., Cares, C., Carvallo, J.P., Grau, G., Haya, M., Salazar, G., Franch, X., Mayol, E., Quer, C.: A Comparative Analysis of i*-Based Goal-Oriented Modelling Languages. In: International Workshop on Agent-Oriented Software Development Methodologies (AOSDM'05), at the SEKE Conference, pp. 657–663 (2005)
4. Basili, V., Caldeira, G., Rombach, H.D.: The Goal Question Metric Approach. Encyclopedia of Software Engineering, Wiley (1994)
5. Basili, V., Rombach, H.: The TAME Project: Towards Improvement Oriented Software Environments. IEEE Transactions on Software Engineering, vol. 14 nº 6, 758–773 (1988)
6. Eclipse: ATL Project, http://www.eclipse.org/m2m/atl/
7. Eclipse: Model Development Tools Project, http://www.eclipse.org/modeling/mdt/
8. Eclipse: UML2 Project, http://www.eclipse.org/uml2/
9. Franch, X.: A Method for the Definition of Metrics over i* Models. In: 21st International Conference on Advanced Information Systems (CAiSE 2009), pp. 201-215. Springer-Verlag LNCS (2009)
10. Franch, X., Grau, G.: Towards a Catalogue of Patterns for Defining Metrics over i* Models. In: 20th International Conference on Advanced Information Systems (CAiSE 2008). LNCS, pp. 197–212. Springer (2008)
11. Genero, M., Piattini, M., Calero, C.: A Survey of Metrics for UML Class Diagrams. Journal of Object Technology, vol. 4 nº 9 (2005)
12. Giachetti, G., Marín, B., Pastor, O.: Using UML as a Domain-Specific Modeling Language: A Proposal for Automatic Generation of UML Profiles. In: CAiSE'09. LNCS (2009)
13. i*: Wiki Web Page, http://istar.rwth-aachen.de/, last Accessed October 2009
14. Kent, S.: Model Driven Engineering. In: Integrated Formal Methods (IFM), pp. 286-298. Springer (2002)
15. Lamsweerde, A.v.: Systematic Requirements Engineering - From System Goals to UML Models to Software Specifications. Wiley (2008)
16. Lucena, M., Santos, E., Silva, M.J., Silva, C., Alencar, F., Castro, J.F.B.: Towards a Unified Metamodel for i*. . In: 2nd IEEE Int. Conference on Research Challenges in Information Science (RCIS 2008), pp. 237–246 IEEE (2008)
17. Marín, B., Giachetti, G., Pastor, O.: Applying a Functional Size Measurement Procedure for Defect Detection in MDD Environments In: 16th European Conference on Systems & Software Process Improvement and Innovation (EuroSPI 2009). CCIS. Springer (2009)
18. Marín, B., Giachetti, G., Pastor, O.: The Photography Agency: A case study of the OO-Method Approach. Technical Report DSIC-II/13/08, Universidad Politécnica de Valencia, Valencia, España (2008)
19. OMG: MOF 2.0 Core Specification
20. OMG: Object Constraint Language 2.0 Specification
21. OMG: UML 2.1.2 Infrastructure Specification
22. OMG: UML 2.1.2 Superstructure Specification
23. OMG: XMI 2.1.1 Specification
24. Pastor, O., Gómez, J., Insfrán, E., Pelechano, V.: The OO-Method Approach for Information Systems Modelling: From Object-Oriented Conceptual Modeling to Automated Programming. In: Information Systems. Elsevier Science, vol. 26 nº 7, 507–534 (2001)
25. Pastor, O., Molina, J.C.: Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling. 1st edition, isbn: 978-3-540-71867-3. Springer, New York (2007)
26. Selic, B.: The Pragmatics of Model-Driven Development. In: IEEE Software, vol. 20 nº 5, 19–25 (2003)
27. Tong, Y., Fangjun, W., Chengzhi, G.: A comparison of metrics for UML class diagrams. ACM SIGSOFT Software Engineering Notes vol. 29 nº 5 (2004)